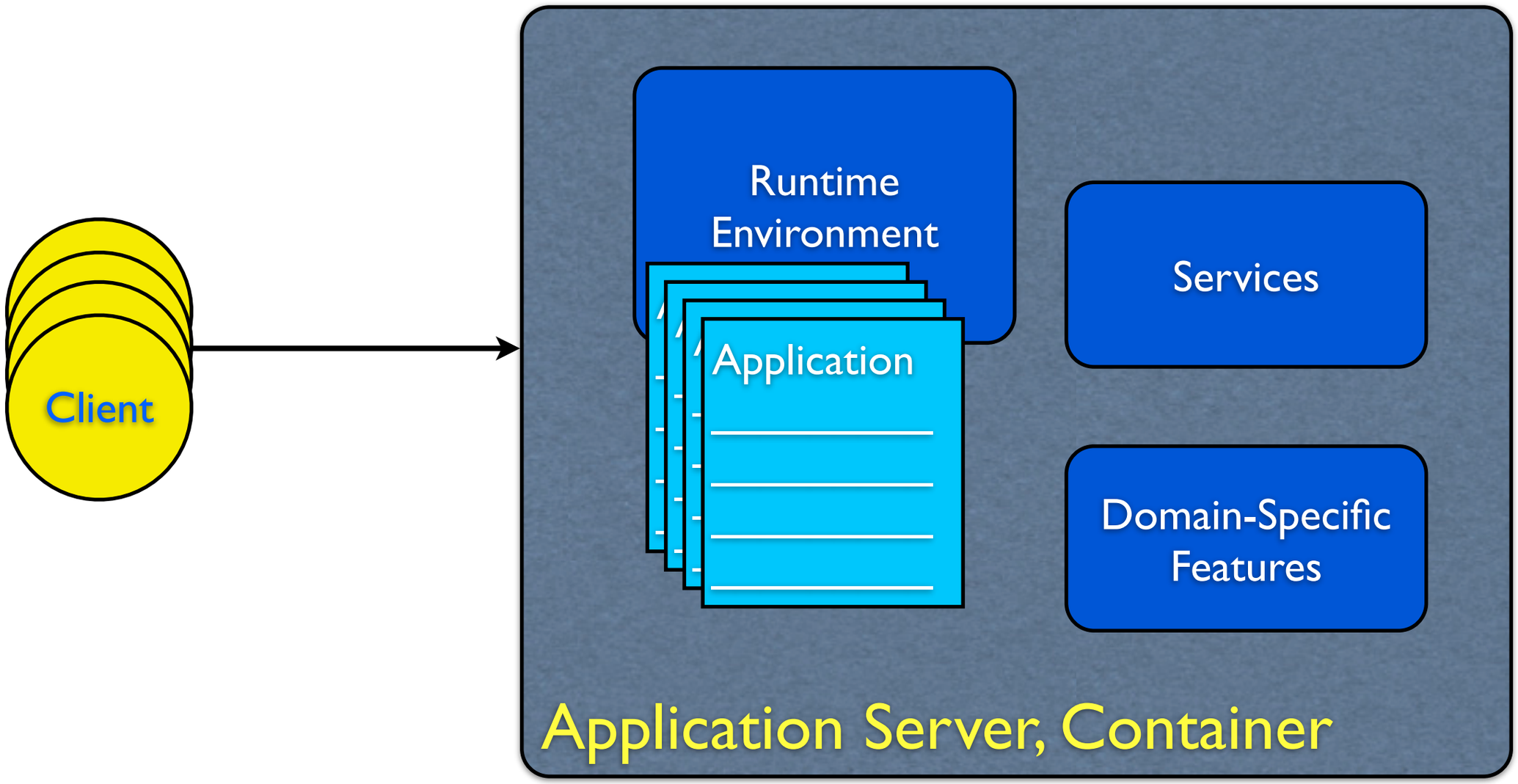
Application Servers

Application Servers



Characteristics

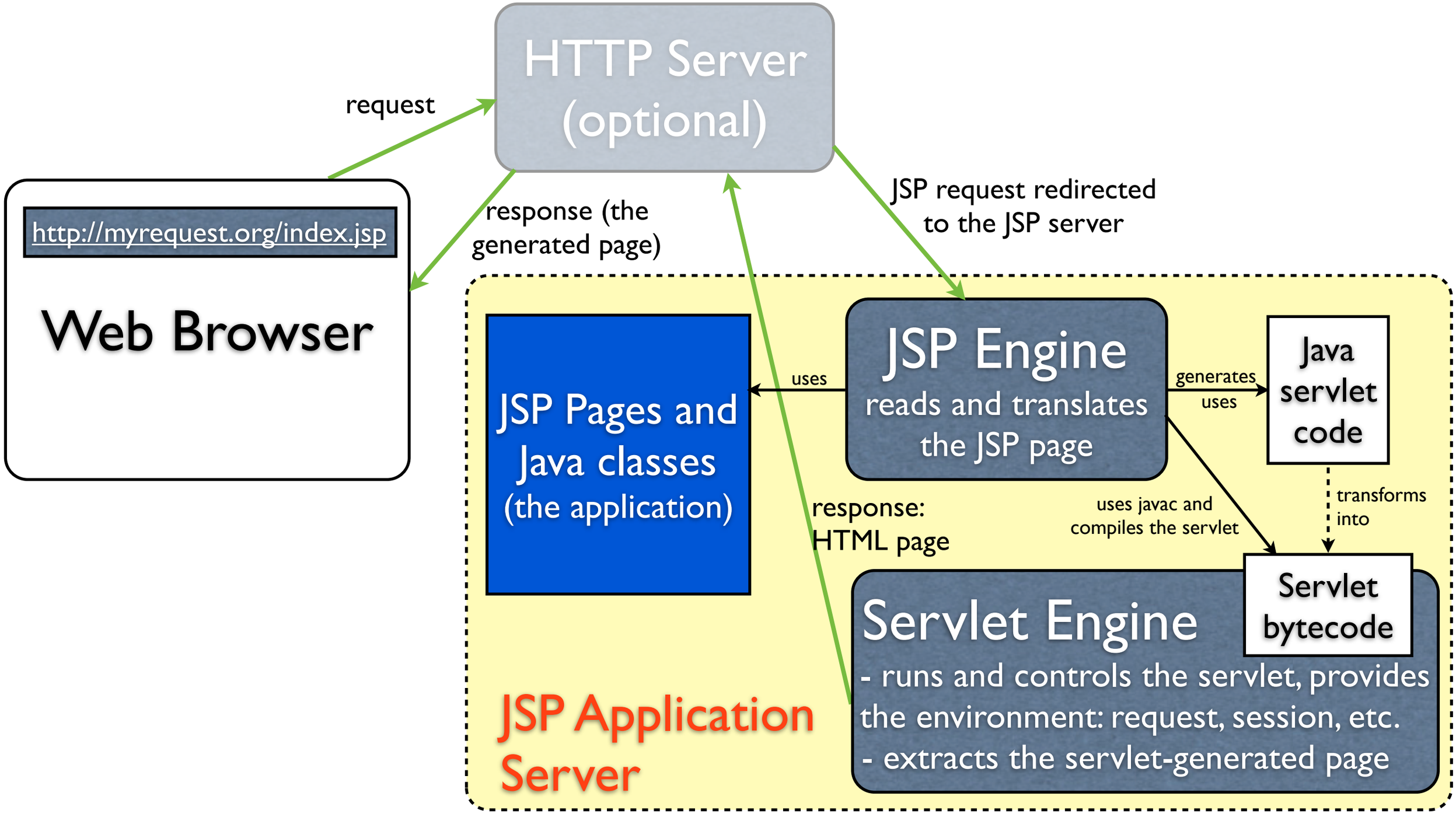
- Applications run *inside* the container
  ~ many applications can be hosted (*deployed*) on the same server
- The main control is at the container
- Applications are provided with a specific environment
- The app server provides complex services, features
  ~ <u>specific to the purpose</u> of the platform
- Not all types of applications are suitable for a particular container-type technology

JSP

JSP

- Java Server Pages
- Technology that builds on Java Servlets
- The container provides support for Web applications
- Easy and quick development of dynamic Web sites

A JSP Application



An Example

- Managing a simple login page in JSP
- Excerpt from a slightly more complex example application [1]

[1] Dan C. Cosma, Programarea aplicatiilor distribuite, Editura de Vest, Timisoara, 2009, ISBN 978-973-36-0501-0

The Main “JSP”

```
<h3>Login</h3>
<form name="login" action="loginAction.jsp" method="post">
<table>
    <tr><td>User Name:</td></tr>
    <tr> <td><input type="text" name="userName"/></td> </tr>
    <tr><td>password:</td> </tr>
    <tr><td>
            <input type="password" name="password"/>
    </td></tr>
    <tr><td align="right">
            <input type="submit" name="add" value="Login"/>
    </td></tr>
</table>
</form>
(c) 2009, Dan Cosma
```

Login

User Name:

password:

Login

(c) 2009, Dan Cosma

The “Action” JSP

```
<\%@ page language="java"
import="java.lang.*,java.util.*" %>
<\%
String userName = request.getParameter("userName");
String password = request.getParameter("password");
if(userName == null)
  { %>
   <p> Please login first.
  <%
  }
else
  { %>
   Congratulations <\%=userName\%>! You are logged in with the password: <\%=password\%>.
<%
  } %>
<p> <hr> <p>
<form name="goMain" action="index.jsp" method="post">
  <input type="submit" name="go" value="Return to main menu"/>
  <!-- Forwarding the username -->
  <input type="hidden" name="userName" value="<\%=request.getParameter("userName")\%>"/>
</form>
```

(c) 2009, Dan Cosma

Comments

- JSP pages mix Java with HTML
  This may lead to unmaintainable code (hard to read and understand)
- JSP pages should be small, and contain as little Java as possible
- Use separate classes for the main Java functionality
  As the JSP will in fact transform into a Java class (the servlet), you can call other classes from within the JSP
- Use a layered model to separate functionalities/ concerns
  In fact, design the program using all the "general" design best practices you are already familiar with

Deployment

- JSPs along with the helper classes are packaged in a standard format (E.g. a *.war* archive with a well-known structure)
- The package is deployed to the container
  This operation is dependent on the chosen JSP application server variant
- The deployment should follow the rules specified by the application server
- The deployment process should be automated
  You should also avoid deploying the application using IDE-specific plugins. The customer doesn't have to install Eclipse to make your program work. Moreover, you don't want to depend too much on a plugin that does "magical" things behind the scene. They will certainly fail you at a point.

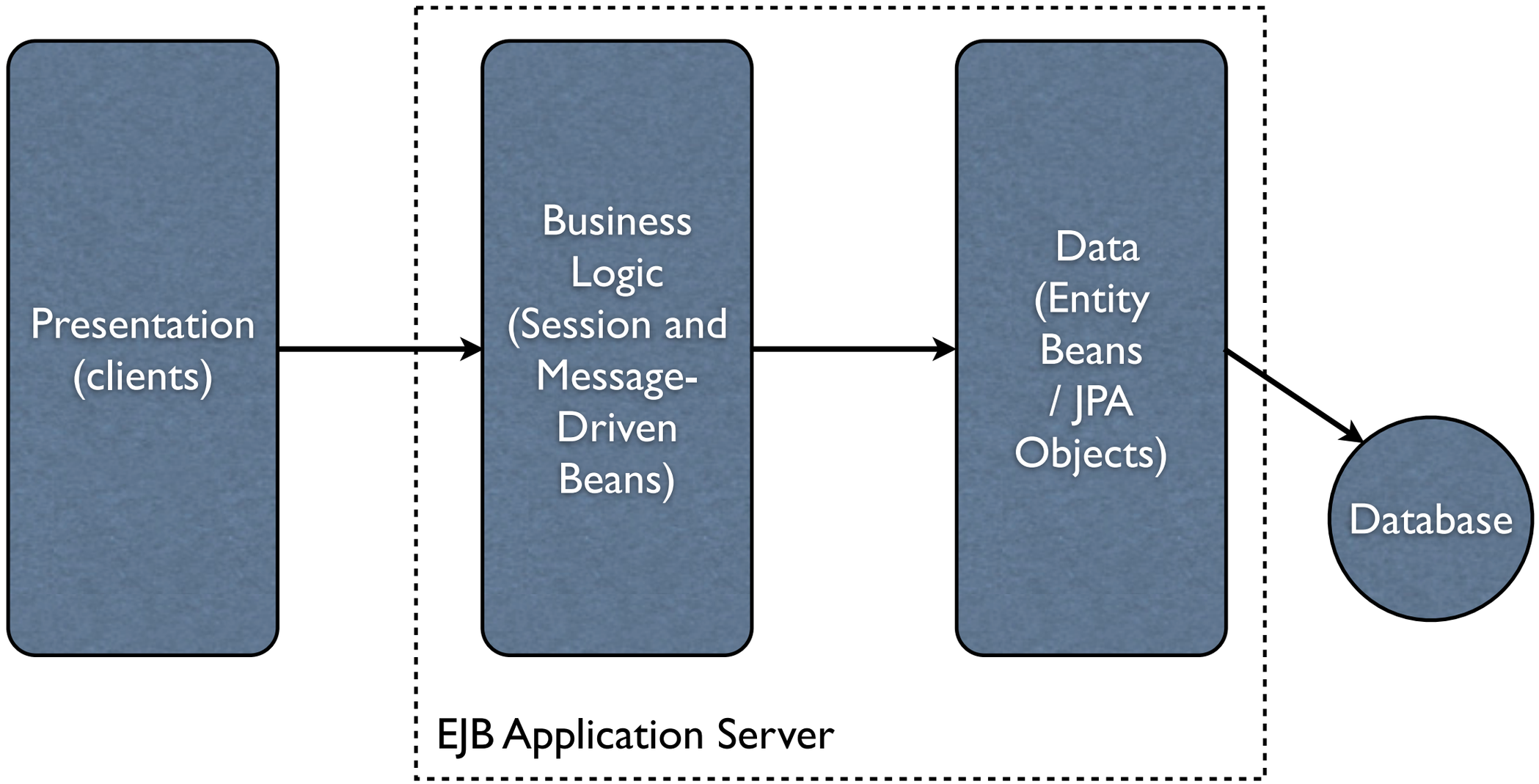
EJB

Enterprise Java Beans

- Part of the J2EE platform
- Provides a framework for building Enterprise applications
- Provides features as: scalability, persistence, clustering, transactions, etc.

Architecture

- EJB applications are three-tier applications



Enterprise Beans

- Entity Beans / Java Persistency API Objects
  - describe the data model
- Session Beans
  - provide the application functionality; they can be either Stateless or Stateful
- Message-Driven Beans
  - participate to the application functionality in a message-driven environment (usually connected with JMS)

An Example

- A simple EJB application: students and courses

The Database

- "student" Table
  - id
  - nume (name)
  - anStudiu (the study year)
- "curs" Table (Course)
  - id
  - nume (name)
  - profesor
- "student_curs" Table (n:n mapping between students and courses)
  - id_student
  - id_curs

JPA Entities

```
package cursuri.jpa;

import java.io.Serializable;
import javax.persistence.*;
import java.util.Set;

/**
 * Persistent class for the student table.
 */

@Entity
@Table(name="student")
public class Student implements Serializable {
        private static final long serialVersionUID = 1
            L;

        @Id
        @GeneratedValue(strategy=GenerationType.AUTO)
        private int id;
```

```
    private int anStudiu;

    @Lob()
    private String nume;

    //bi-directional many-to-many association to
        Curs
    @ManyToMany
    @JoinTable(
            name="student_curs"
            , joinColumns={
                    @JoinColumn(name="id_student")
                    }
            , inverseJoinColumns={
                    @JoinColumn(name="id_curs")
                    }
            )
    private Set<Curs> curs;
```

```java
public Student() {
}

    public int getId() {
            return this.id;
    }

    public void setId(int id) {
            this.id = id;
    }

    public int getAnStudiu() {
            return this.anStudiu;
    }

    public void setAnStudiu(int anStudiu) {
            this.anStudiu = anStudiu;
    }
```

```java
            public String getNume() {
                    return this.nume;
            }

            public void setNume(String nume) {
                    this.nume = nume;
            }

            public Set<Curs> getCurs() {
                    return this.curs;
            }

            public void setCurs(Set<Curs> curs) {
                    this.curs = curs;
            }

    }
```

```java
package cursuri.jpa;

import java.io.Serializable;
import javax.persistence.*;
import java.util.Set;



/**
 * The persistent class for the curs database table.
 */
@Entity
@Table(name="curs")
@NamedQuery(name="Curs.getByName", query="SELECT c
    FROM Curs c WHERE c.nume LIKE :text")
public class Curs implements Serializable {
        private static final long serialVersionUID = 1
            L;

        @Id
        @GeneratedValue(strategy=GenerationType.AUTO)
        private int id;

    @Lob()
        private String nume;

    @Lob()
        private String profesor;
```

```java
    //bi-directional many-to-many association to
        Student
    @ManyToMany(mappedBy="curs")
    private Set<Student> students;

public Curs() {
}

    public int getId() {
            return this.id;
    }

    public void setId(int id) {
            this.id = id;
    }

    public String getNume() {
            return this.nume;
    }
```

```java
public void setNume(String nume) {
        this.nume = nume;
}

public String getProfesor() {
        return this.profesor;
}

public void setProfesor(String profesor) {
        this.profesor = profesor;
}

public Set<Student> getStudents() {
        return this.students;
}

public void setStudents(Set<Student> students)
    {
        this.students = students;
}

}
```

Session: Remote, Local

```java
package cursuri.ejb;
import java.rmi.RemoteException;
import java.util.ArrayList;

import javax.ejb.Remote;

@Remote
public interface InscrieRemote {
        public ArrayList<CursInfo> cursuri(String
            tipar) throws RemoteException;
        public void inscrieStudent(String nume, int
            cursId) throws RemoteException;
}
```

```java
package cursuri.ejb;
import java.util.ArrayList;

import javax.ejb.Local;

@Local
public interface InscrieLocal {
        public ArrayList<CursInfo> cursuri(String
            tipar);
        public void inscrieStudent(String nume, int
            cursId);
}
```

Session: Implementation

```
package cursuri.ejb;

import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.List;


import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import cursuri.jpa.Curs;

/**
 * Session Bean implementation class Inscrie
 */
@Stateless(name="Inscrie", mappedName="ejb/InscrieJNDI
    ")
public class Inscrie implements InscrieRemote {
```

```java
    @PersistenceContext
    private EntityManager entityManager;

/**
 * Default constructor.
 */
public Inscrie() {
    // TODO Auto-generated constructor stub
}
public ArrayList<CursInfo> cursuri(String
    tipar) throws RemoteException {

        ArrayList<CursInfo> lista = new
            ArrayList<CursInfo>();
        List objList = (List)entityManager.
            createNamedQuery("Curs.getByName").
            setParameter("text", tipar).
            getResultList();
        for(Object item : objList)
        {
                lista.add(new CursInfo(((Curs)
                    item).getId(), ((Curs)item)
                    .getNume().toString()));
        }
        return lista;
}

public void inscrieStudent(String nume, int
    cursId) throws RemoteException {

                ...
}}
```

Client code (excerpt)

```
InitialContext ctx = null;
try {
        ctx = new InitialContext();
} catch (NamingException e) {
        e.printStackTrace();
}
InscrieRemote bean = null;
try {
        bean = (InscrieRemote) ctx.lookup("ejb/
            InscrieJNDI");
} catch (NamingException e) {
        e.printStackTrace();
}

ArrayList<CursInfo> lista = null;
try {
        lista = bean.cursuri(tipar);
} catch (RemoteException e) {
        e.printStackTrace();
}
```

Web Services

Web Services

- Definition (W3C):

  ”A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.”

Source: http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/

What Is a Web Service?

- A component of a distributed application:
  - self contained and self described
  - accessible through the network
  - communicating using standardized protocols
  - discoverable by other parties through various methods
  - data exchange format is usually XML

Web Services

- Types of Web services:
  - arbitrary Web services, in which the service may expose an arbitrary set of operations
  - REST-compliant Web services, in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of "stateless" operations; and

Source: http://www.w3.org/TR/ws-arch/#relwwwrest

Technologies

- SOAP - the communication protocol
- WSDL - the service description
- UDDI - the service discovery
- XML, JSON - data format

SOAP

- The current development of SOAP defines two acronym expansions:
  - Simple Object Access Protocol - a message represents a remote method invocation (using the SOAP RPC representation)
  - Service-Oriented Architecture Protocol - the message represents the information passed to/from a service in a loosely-coupled, message-based, service architecture

SOAP

- Defines the communication protocol and XML data formats for exchanging messages
- The structure of a SOAP message:
  - Envelope - identifies the XML as a SOAP message
  - Header - application-specific
  - Body - invocation or response information
  - Fault - errors, status information

WSDL

- Web Services Description Language
  - XML-based
  - Describes the Web service as a collection of operations (methods) exposed publicly
- The WSDL 2.0 document elements:
  - Service - the container
  - Endpoint - service location (e.g. an URL)
  - Binding - specifies the interface and the SOAP binding style (Document/RPC)
  - Interface - defines the service, its operations and messages
  - Operation - the exposed methods
  - Types - the type of the data

“Classic” Web Services

- Operations are specified freely, as application-specific constructs (e.g. “getCustomerData”)
- Services may be located through UDDI nodes
- Services are described through WSDL
- SOAP is used as RPC or as message-orientation support

REST

- Representational State Transfer
- An architectural style suited for the Web
- Introduced by Roy Fielding in 2000
  http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

REST

- The REST philosophy:
  - Design a Web service focusing on system *resources*
  - A resource is identified by an URI (Uniform Resource Identifier)
  - A representation of a resource is a document capturing the state of the resource
  - Clients and servers exchange resource representations
  - Client requests to the servers are made when a transition to a new state is required
  - REST services are stateless

REST

- Defines a set of architectural principles for Web services:
  - Use HTTP methods as they were designed
  - The service is stateless
  - Resources are organized in a directory-like structure of URIs
  - Transfer XML, JSON (JavaScript Object Notation), or both

Source: https://www.ibm.com/developerworks/webservices/library/ws-restful/

Use HTTP Methods

- REST applications directly map their operations on the standard HTTP methods:
  - To create a resource on the server: POST
  - To retrieve a resource: GET
  - To change the state of a resource: PUT
  - To delete a resource: DELETE

- Example: instead of GET /adduser?name=Robert HTTP/1.1 use

```
POST /users HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>  <name>Robert</name> </user>
```

Source: https://www.ibm.com/developerworks/webservices/library/ws-restful/

REST is Stateless

- The service does not store state information
- When making a request, the client presents the server with all the necessary state information so that the request can be fulfilled
- Improves scalability, simplifies the design

Source: https://www.ibm.com/developerworks/webservices/library/ws-restful/

REST: URIs as Directories

- Resources should be represented analogous to a directory structure
- The URIs should be as simple and intuitive as possible, should be lowercase-only
- Examples:
  http://www.myservice.org/discussion/topics/computers
  http://www.myservice.org/discussion/topics/science/threads
  http://www.myservice.org/discussion/{year}/{day}/{month}/{topic}

Source: https://www.ibm.com/developerworks/webservices/library/ws-restful/

REST: Data Transfer

- The representations of the resources represent the resource state and attributes (a “snapshot” in time for that specific resource)
- Clients receive the representations upon request
- They can use XML, JSON or other structured formats

Source: https://www.ibm.com/developerworks/webservices/library/ws-restful/